

Meta-Score, a Novel PWGL Editor Designed for the Structural, Temporal, and Procedural Description of a Musical Composition

Mika Kuuskankare

The DocMus Department of Doctoral Studies in Musical Performance and Research
Sibelius Academy, Finland

Abstract

In this paper we introduce a prototype of 'meta-score', a novel visual editor in PWGL, aimed at defining the structural, temporal and procedural properties of a musical composition.

Meta-score is a music notation editor, thus, the score can be created manually by inputting the information using a GUI. However, meta-score extends the concept of a musical score so that the musical content can be defined not only manually but also procedurally. The composition is defined by placing scores (hence the name meta-score) on a timeline, creating dependencies between the objects, and defining the compositional processes associated with them.

Meta-score presents the users with a three-stage compositional process beginning from the sketching of the overall structure along with the associated harmonic, rhythmic and melodic material; continuing with the procedural description of the composition and ending with the automatic production of the performance score.

In this paper, we describe the present state of meta-score.

Background

The compositional process is divided into several steps, which include, among others, finding and exploring the base material, planning the overall form, sketching the individual passages, and transcribing the material into the final score. Usually, only some of the steps are realized with the help of a computer. The composer might use Computer-assisted Composition (CAC) software to explore the base material but do the sketching using a pen and paper. Often composers choose to do the higher-level structural planning using pen and paper because it's easier to experiment, add, remove, and rearrange material. Furthermore, the computerized steps can potentially be realized using a different piece of software.

Most CAC software lacks a real score representation that would allow for the manipulation of all the properties of the score such as instrumentation, articulations, or even arbitrary graphics, as a part of the compositional process.

The PWGL(Laurson, Kuuskankare, and Norilo 2009) environment provides users with music notational capabilities through its notational front-end, ENP(Kuuskankare and Laurson 2006). ENP allows us to represent complex musical output, consisting not only of rhythm and pitch, but also of expressions, arbitrary user-definable graphics, etc. However, we currently lack a compositional/notational tool that would allow us to easily manipulate the complex structural organization of a musical composition. To that end we have implemented a novel ENP-based editor, called meta-score. It allows us to arrange score objects against a global sequence of time-signatures on a canvas representing the composition. It is possible to define the properties of the objects, and associate them with computational processes thus defining the composition. Meta-score attempts to provide composers with an environment in which it is possible to realize the whole compositional process: working with the base material, defining the structural and procedural organization of the composition, and automatically producing the final score.

There are several important concepts behind meta-score: First, being based on ENP, it lets users manipulate the musical objects directly by editing the pitch and rhythm, inserting expressions, and defining the instrumentation. Thus it behaves like a real score editor. Second, each object has its own temporal structure, instrumentation and articulations. Further to that, they can be written using either rhythm notation or time notation. Thus, the objects can be manipulated independently of the global structure. Third, meta-score extends the concept of a musical score so that the musical content can be defined not only manually but also procedurally. The objects can have both temporal and procedural dependencies and they can have generative processes associated with them. Fourth, it is possible to visually define a global structure (temporarily organized base material) using real musical objects. The global objects can be manipulated and generated in the same way as other types of objects. Finally, it provides strict visual/temporal synchronization allowing the temporal organization and rearrangement of both the score objects and the base material.

Previous work in this area consists primarily of the work done at IRCAM in the form of Maquette and Sheets (J. Bresson 2011). Maquette is an extension of an OpenMusic (Assayag et al. 1999) patch with a time dimension. Sheets is

another attempt to develop a compositional tool in Open-Music. However, as opposed to meta-score, both Maquette and Sheets are not score editors strictly speaking but instead they both extend the concept of a visual patch. The notation is not directly editable at the score level, the score representation does not contain higher level structures such as parts or instruments, and there is no concept of expressions. Also, in Maquette, there is no strict visual synchronization.

The main objective in the meta-score project is to study the possibilities of automation in the compositional process using a rich music notation representation as a starting point. Currently, there is no comparable environment. Most music notation software does not allow for the algorithmic generation of material. Furthermore, they are not well suited for sketching because of their rigid, and non-modular music representation models. The available CAC environments usually lack full music representation concentrating mainly on representing pitch and rhythm. This will make it impossible to write, for example, idiomatic instrumental music. Moreover, it will make it impossible to include, in the compositional process, more unconventional components, such as electro-acoustic parts.

The meta-score is still in a prototypical stage, and under active development. Eventually, it will be part of the family of musical editors found inside PWGL.

Meta-score

The fundamental idea behind meta-score is somewhat similar to that of L^AT_EX, the document preparation system used widely in academia, that automates most aspects of typesetting and desktop publishing. Similarly, when working with meta-score, composers work on the description of the composition, defining the base material, the processes, and the temporal organization. The product, the performers score, is produced automatically from that description.

At the moment, the score is generated with the help of Lilypond, mainly because we want to experiment with the automatic production of all of the components of the score. Furthermore, the use of Lilypond allows us to export a rich set of special notational graphics, such as microtones, tape notation, time notation, etc. In the future, the final score will be generated with the help of ENP.

The user is not assumed to directly edit the final product. Although this is perfectly possible, as the output is a complete Lilypond project, it makes little sense in our case since it breaks the connection between the source and the end product. Instead, if a change is needed, the composer/user goes back to the description, makes the changes, and generates the final score again. This approach is very much along the lines of some of the pioneers of algorithmic composition, namely Lejaren Hiller and Pierre Barbaud, who believed that:

[...] the output score generated by a composition program should not be edited by hand; rather, the program logic should be changed and run again.

(Roads 1996)

Components of Meta-score

The structural definition consists of the global structure, the temporal structure (time signatures), and the score objects. The logical definition consists of the dependencies and the processes. The three parts of the global structure are described first, and are then followed by the descriptions of the various system dependencies. The sections to follow outline the details of the process description that can be used to manipulate objects.

Global Structure

Global structure allows us to represent base material that can be aligned with musical objects. It can be used to define the higher-level structural organization of the composition, such as, the overall harmonic plan. We can also represent other time-varying information, such as, breakpoint functions. The latter could be used to visually define the overall 'tension' or to define a global dynamics curve.

The global objects are drawn above the score. Visually, they can be drawn either as empty rectangles indicating only their temporal extent, or they can reflect the material they represent by displaying a sequence of chords, a breakpoint function, etc.

Temporal Structure

The temporal structure is defined by a stream of time-signatures (see Figure 1). The time-signatures are superimposed on top of the score and they always override the local structures. This makes it convenient to 'restructure' musical objects. When creating the final score the objects are quantized according to global time-signatures.

The Objects

The musical objects that can be arranged along the timeline are ENP scores with their own internal structure and attributes. Any pre-existing score can be pasted or imported into meta-score. Their content can be directly edited and they can be freely positioned both vertically and horizontally. The horizontal placement defines the start time of the scores. The vertical placement is purely visual and doesn't have any functional purpose.

Dependencies

Each object can depend on another object. Currently, an object can only depend on one other object at a time. Circular dependencies are not allowed. A dependency is visualized by drawing an arrow between the objects, with the arrow head pointing towards the depending object (see Figure 1b for an example). From the UI point of view a dependency can be made by dragging a connection line from one object to another. When the connection is made the programs asks the user which type of dependency he/she wants to make. The available dependency types are enumerated in the subsections that follow.

Procedural Dependencies A procedural dependency can be created between two objects. A connection creates a dependency with two main purposes. It provides the processes

with input material and it defines the order of calculation. As an example, if object A depends on object B, the contents of B are used when composing object A. Therefore, B has to be composed before A can access its contents. To ensure the correct order of calculation the scores are topologically sorted. The processes are then applied to each of the scores in the given order.

Temporal Dependencies It is also possible to define temporal dependencies. The start-time of an object can depend on the start-time of another object. For example, a score could be 'pinned' to a time-signature and would thus get its temporal cues from there. Furthermore, an object can be constrained to start only after another object has ended, thus creating sequences of objects.

Figure 1 shows both kinds of dependencies:

- (a) and (b) are procedural dependencies. The object marked as 'Vibraphone' reads the pitch content of the 'Flute' and creates the displayed chord according to a process defined by the user. The connection (b) connects the two 'Flute' objects. The process of the second object is 'copy', which creates an exact copy of the dependency.
- (c) is a temporal dependency, i.e., the start-time of the object depends on the start-time of the time-signature to which it is connected. The small icon resembling a clock face is used here to mark a temporal dependency.

Logical Dependencies As will be explained in the section 'The Final Output', meta-score constructs the final score automatically based on the definition provided by the user. This includes deciding the number of instruments needed to realize the score. However, the user can provide hints as to how to connect different objects in the final stage. Normally, the vertical arrangement of the objects is of no importance, but, by defining logical dependencies between objects they can be grouped as one 'vertical line'.

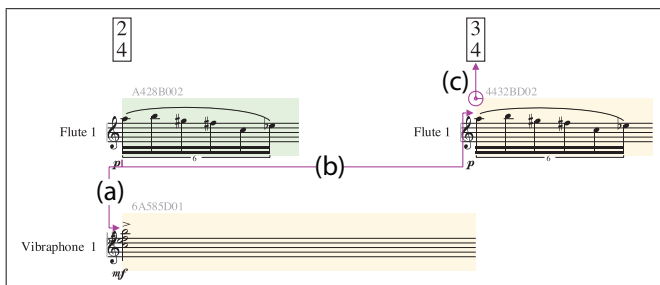


Figure 1: Procedural dependencies (a) and (b), and temporal dependency (c).

Cloning Musical Material In addition to all of the possibilities of defining musical material through processes, the meta-score allows the users to clone musical material by using a special kind of dependency. A 'cloning' dependency makes an exact 'live' copy of another object. The object can be displaced in time but its musical material will always reflect that of its data source.

Process Description

Each of the objects can be associated with a process description, defining both the name of the actual process as well as all the properties associated with it. Currently, the processes are defined as Lisp functions. In the future, they may be defined using a visual patch.

The input for the process is a score. The process either destructively manipulates its musical content or replaces it with the result of the process. The return value must be either the processed score itself or a new score. In the former case the input score is manipulated destructively by the process, and, in the latter case the contents of the processed score are substituted with the new ones. Thus, the meta-score processes are functions that use a score both as a parameter and as a return value.

In addition to 'process', the objects can contain information about instrumentation, scripts, rules, articulation, dynamics, etc. We have implemented a small domain-specific language for entering the attributes for the objects. The language is based on the concept of key-value pairs; thus, the process description is a collection of tuples. An open-ended data structure was chosen because it allows attributes to be added and removed easily.

Most of the attributes are user-definable, i.e., their use and behavior depends on the associated process. However, there are some predefined attributes, such as articulation and dynamics, that behave more like post processes, i.e., they apply a certain operation to the score after it has been generated or processed:

1. `:instrument (s)`

This attribute defines the collection of instruments that can be used by the given object. The instruments can be entered using a 'standard' syntax understood by professional musicians, e.g., `fl, ob, 2 cl`, for flute, oboe, and two clarinets.

2. `:articulation`


The attributes are defined using common names, such as `accent`, `staccato`, etc. Any number of articulations can be given.

3. `:playing-style`

defines an instrument-specific playing style applied to the score, for example, `'pizzicato'` or `'sul-tasto'`.

4. `:dynamics`

The dynamics are defined by the tuple `:dynamics <value>`, e.g., `:dynamics pp`. In addition to single dynamic markings, several shorthands are provided for some of the most common combinations. For example,

the shorthand `o<mp` means  `mp`, i.e., a crescendo that starts from total silence (niente) and ends in mezzo-piano.

Moreover, the attributes `:pre-process` and `:post-process` are used to name a list of operations applied to the score before and after calculation.

Example 1 gives one possible process definition for an object whose musical content is generated using constraints

(the process name is 'csp', as in Constraints Satisfaction Problem). The definition also shows a set of rules (the :rules attribute) as well as both the :dynamics and :articulation attributes.

```

:process csp
:rules ((+ ?1 ?2 (?if (< (abs (- (m ?2) (m ?1))) 5))))
:dynamics o<mp
:articulation sul-tasto

```

Example 1: A meta-score object attribute definition. The process is 'csp', i.e., a constraint process. The rules are given using the keyword :rules.

A Meta-score Example

In this section we will present a brief compositional assignment defined with the help of meta-score. (See Figure 2).

We begin our example by composing the melodic line by hand for the bassoon shown in (a). Next, our compositional idea is to produce a background texture for the bassoon. To accomplish this we add the object shown in (b).¹

As our next step, we define the process of the object (b) as :pedal. Pedal objects create orchestral pedal texture according to the pitch and rhythm received from the object on which it depends. The instruments used by the pedal object can be defined by the :instrument key with a value, such as, :strings or :woodwinds. The pedal process needs a data-source in order to operate, thus, we create the connection (c) between the objects (a) and (b), i.e., we define that object (b) depends on object (a). The boundaries (vertical and horizontal) of the object are adjusted accordingly as soon as the process is applied.

Next, we write the two lines for French Horn shown as (d) and (e). The object (f) is created automatically by connecting (e) to (f) using the connection (g). The process of (f) is :copy, which means that an exact copy is made of the dependency.

Finally, an object with another kind of process can be seen in (h). This object, in turn, uses (b) as a data-source, through the connection (i), and its process is :accent. The object looks for the pitch content of its dependency and generates a chord, a percussive attack, reinforcing the current harmony.² The final score can be seen in Figure 3.

A list of all of the objects in topological order is displayed in the sidebar (see Figure 2). Here, the user can see the order of calculation and the dependencies, and can go to a given object by clicking its name in the list. By default, the objects are represented by their unique, automatically generated ID numbers, for example the object with ID# 66976595 is the bassoon (a) and the ID# 66970694 represents the strings shown in (b). However, even as these identification numbers

¹By default, the objects do not contain any data or have a duration. Their vertical ordering is not important and can be freely adjusted by the user.

²Here, the instrument is Vibraphone. The generative objects, such as :accent and :pedal also take into consideration the instrument ranges.

are generated automatically by the system, the user is free to rename the objects.

The Final Output

The final output is the performers' score exemplified in Figure 3. Meta-score also generates an instrumentation page containing information about the needed instruments, doublings, and range when appropriate. This would also include percussion pictograms if the composer chooses to use them instead of textual indicators. Figure 4 shows a potential instrumentation page.

Generating the Parts

The objects are automatically combined into parts. Every object using a specific instrument is collected into a pool of objects. In the simplest case, if there are no overlapping events, one part is created for that particular instrument. If there are overlapping events the objects are arranged so that they result in a minimum number of parts.

Currently, the algorithm we use is simple and straightforward. For each instrument, there are multiple tracks (i.e., possible parts) where individual objects can be assigned. The objects are first sorted by time and pitch (the mean). After this the algorithm assigns each object to one of the tracks starting always from the first track. In case the object cannot be fit in the current track, the next one is tested until one with sufficient free space is found.

In our example, the original meta-score definition contained six violin parts. However, it is not defined which of the two violin sections is going to play the parts. This is not an oversight but rather by design. The parts needed to perform a certain score are constructed by the meta-score editor. We are only concerned by the number of simultaneous parts. Here, after having determined that there are six simultaneous parts written for the 'same' instrument, the exporter uses a set of rules as to how to divide the instruments reasonably among the available instrument sections, as can be seen in Figure 3. The same applies to the horn part. In the final score there is only one horn (as can be seen in Figure 3) since the horn lines in the meta-score do not overlap.

In the future there will be more heuristics involved, such as assigning certain kinds of passages to certain 'members' of the section. For example, assigning the objects for certain parts according to their rhythm or pitch content (faster/higher) or according to their additional instrument content. As an example, the higher and more agile parts would be assigned to the 1st player, or, when a piccolo is called for in an object written for the flute part, the object would be assigned to the 2nd player. Likewise, objects containing the indication 'solo' would be assigned to the 1st players.

Discussion

The current prototype of meta-score is fully functional. The scores can be manipulated both algorithmically and manually. The dependency scheme is implemented allowing a single dependency per object. Currently, the processes can be defined using Lisp code and there is a collection of 'house

Figure 2: The meta-score editor with both hand written and automatically generated material (dependencies between the objects indicated by the arrow-headed connections).

TITLE
sub title

COMPOSER

$\text{♩} = 60$

Bassoon

French Horn 1

Vibraphone

Violin I

Violin II

pp

pp

pp

pp

pp

pp

Figure 3: The final score, generated from the meta-score description shown in Figure 2, typeset using Lilypond.

INSTRUMENTS

2 Flutes (1st doubling Piccolo ; 2nd doubling Alto Flute)
 Oboe (doubling English Horn)
 *Percussion (6)
 Timpani
 Piano

*1) Marimba, Agogo Bells, Xylophone, 4 Tom -Toms, Crash Cymbals (60"), Gong (60")


2) 4 Temple Blocks, Mark Tree, Gong (60")

3) Mark Tree, 2 Suspended Cymbals (medium and very large), Wood Block

4) Triangle, Vibraphone, Xylophone, Tam -tam (47"), Maracas

5) Flex -A-Tone

5) 6 Thai Gongs



6) 6 Crotales




Figure 4: The instrumentation page generated automatically by the meta-score.

processes' already defined, including the pedal process used in this paper. The final score is produced automatically. However, there are only a handful of orchestration methods defined.

Several interesting problems remain to be investigated further: First, we should work on the temporal and procedural dependency scheme. The objects should be able to have multiple dependencies.

Second, it would be interesting to investigate the use of circular dependencies. One could imagine connecting the last object in the score back to the first one thus creating a feedback loop (a kind of evolving score). The composer could then step through different evolutions of the score.

Third, converting the composer's score into a performer's score presents several challenges: how to divide the fragments between instruments, how many instruments to use, and how to inform the composer about decisions or constraints that are impractical. This is an interesting problem as such because it touches many important aspects of composition such as orchestration and instrument idiomatic writing. Computationally, this task could probably be seen as a Resource Minimization Job Scheduling problem. More formally, given a set P of score fragments for a given instrument, where each score fragment p is associated with an onset time op , duration dp , and user-definable preferences pp (such as range or agility), our goal is to allocate all score fragments to specific instrument parts, using the minimum possible number of instruments.

Fourth, we should investigate different strategies for representing the structural organization of the composition. It could be represented hierarchically, for example, as a tree. This would, at the same time, act as a structural description and a user-interface component allowing us to select, add, remove and rearrange sections or a selection thereof. The selected sections, could, then, be displayed and edited on the meta-score canvas. At the least, the score could be divided into 'scenes' much like in the case of video-editing

software. These 'scenes'—or 'sections' in our case—could be displayed on top of the time-line. They could have their own time-line so that their horizontal size would correspond to their proportional duration.

Finally, as a computational improvement, we should divide the problem into sub-problems that could potentially be solved in parallel. To that end, we should move to a more powerful topological sorting algorithm, such as that of Tarjan' (Tarjan 1972).

Acknowledgment

The work of Mika Kuuskankare has been supported by the Academy of Finland (SA137619). The author would also like to thank CCRMA for hosting this research.

References

- Assayag, G.; Rueda, C.; Laurson, M.; Agon, C.; and Delerue, O. 1999. Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal* 23(3):59–72.
- J. Bresson, C. A. 2011. Visual programming and music score generation with openmusic. In *IEEE Symposium on Visual Languages and Human-Centric Computing*.
- Kuuskankare, M., and Laurson, M. 2006. Expressive Notation Package. *Computer Music Journal* 30(4):67–79.
- Laurson, M.; Kuuskankare, M.; and Norilo, V. 2009. An Overview of PWGL, a Visual Programming Environment for Music. *Computer Music Journal* 33(1):19–31.
- Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, Massachusetts, London, England: The MIT Press.
- Tarjan, R. E. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1(2):146–160.